

# Още итеративни алгоритми

Тодор Дуков

## Подход при задачи с вече даден алгоритъм

Нека разгледаме следния алгоритъм:

```
1  int foo(int a)
2  {
3      int x = 6, y = 1, z = 0;
4
5      for (int i = 0; i < a; ++i)
6      {
7          z += y;
8          y += x;
9          x += 6;
10     }
11
12     return z;
13 }
```

Питаме се какво връща той?

Обикновено в такива задачи трябва да се изпробва алгоритъма върху няколко стойности. Можем да забележим, че:

- $\text{foo}(0)$  връща 0
- $\text{foo}(1)$  връща 1
- $\text{foo}(2)$  връща 8
- $\text{foo}(3)$  връща 27
- $\text{foo}(4)$  връща 64
- $\text{foo}(5)$  връща 125
- $\text{foo}(6)$  връща 216
- $\text{foo}(7)$  връща 343
- $\text{foo}(8)$  връща 512

Вече можем да видим какво прави алгоритъмът –  $\text{foo}(a)$  връща  $a^3$ . Нека сега докажем това:

**Инвариант.** При всяко достигане на проверката за край на цикъла на ред 5 имаме, че:

- $x = 6(1 + i)$
- $y = 3i^2 + 3i + 1$
- $z = i^3$

**База.** При първото достигане имаме, че:

- $i = 0$
- $x = 6 = 6 \cdot 1 = 6(1 + 0) = 6(1 + i)$
- $y = 1 = 3 \cdot 0^2 + 3 \cdot 0 + 1 = 3i^2 + 3i + 1$
- $z = 0 = 0^3 = i^3$

**Поддръжка.** Нека твърдението е изпълнено за някое непоследно достигане на проверката за край на цикъла. Тогава при влизане в тялото на цикъла:

- $z$  ще стане  $z + y \stackrel{(\text{ИП})}{=} i^3 + 3i^2 + 3i + 1 = \underbrace{(i+1)^3}_{\text{ново } i}$
- $y$  ще стане  $y + x \stackrel{(\text{ИП})}{=} 3i^2 + 3i + 1 + 6 + 6i = 3\underbrace{(i+1)^2}_{\text{ново } i} + 3\underbrace{(i+1)}_{\text{ново } i} + 1$
- $x$  ще стане  $x + 6 \stackrel{(\text{ИП})}{=} 6(1+i) + 6 = 6(1 + \underbrace{i+1}_{\text{ново } i})$

**Терминация.** В последното достигане на проверката за край на цикъла имаме, че  $i = a$ , и тогава на ред 12 алгоритъмът ще върне  $z = a^3$ .

Нека разгледаме още един такъв пример:

```

1  int bar(int n)
2  {
3      return (int)sqrt((double)n * n);
4  }
5
6  int foo(int x, int y)
7  {
8      return (x + y + bar(x - y)) / 2;
9  }
10
11 int quux(int *arr, int n)
12 {
13     int a = arr[0];
14
15     for (int i = 1; i < n; ++i)
16     {
17         a = foo(a, arr[i]);
18     }
19
20     return a;
21 }
```

Искаме да видим какво връща `quux(arr, n)` където `arr` е масив от цели числа а `n` е броят на неговите елементи. Тук най-трудното, което трябва да се направи, е да се определи какво връщат `bar` и `foo`:

- $\text{bar}(n) = \sqrt{n^2} = |n|$ :
  - ако  $n \geq 0$ , то  $\sqrt{n^2} = n = |n|$
  - ако  $n < 0$ , то  $\sqrt{n^2} = -n = |n|$
- $\text{foo}(x, y) = \frac{x+y+|x-y|}{2} = \max\{x, y\}$ :
  - ако  $x \geq y$ , то  $\frac{x+y+|x-y|}{2} = \frac{x+y+x-y}{2} = \frac{2x}{2} = x = \max\{x, y\}$
  - ако  $x < y$ , то  $\frac{x+y+|x-y|}{2} = \frac{x+y+y-x}{2} = \frac{2y}{2} = y = \max\{x, y\}$

Като знаем какво правят `bar` и `foo`, можем да забележим, че `quux(arr, n)` дава най-големия елемент на `arr`:

**Инвариант.** При всяко достигане на проверката за край на цикъла на ред 15 имаме, че  $a = \max \text{arr}[0 \dots i - 1]$ .

**База.** При първото достигане имаме, че  $a = \text{arr}[0] = \max \text{arr}[0 \dots 1 - 1] = \max \text{arr}[0 \dots i - 1]$ .

**Поддръжка.** Нека твърдението е изпълнено за някое непоследно достигане на проверката за край на цикъла. Тогава като влезем в тялото на цикъла, променливата `a` става:

$$\begin{aligned} \text{foo}(a, \text{arr}[i]) &\stackrel{(\text{ИП})}{=} \text{foo}(\max \text{arr}[0 \dots i - 1], \text{arr}[i]) = \max(\max \text{arr}[0 \dots i - 1], \text{arr}[i]) \\ &= \max \text{arr}[0 \dots i] = \max \text{arr}[i \dots \underbrace{i+1-1}_{\text{ново } i}] \end{aligned}$$

**Терминация.** При последното достигане на проверката за край на цикъла на ред 15 променливата `i` ще бъде `n`, откъдето функцията ще върне точно  $a = \max \text{arr}[0 \dots n - 1]$ , с което сме готови.

## Задачи

Задача 1. Даден е следният алгоритъм:

```
1 int num_slopes(int *arr, int n)
2 {
3     int slopes = 1;
4
5     for (int i = 1; i < n; ++i)
6     {
7         if (arr[i - 1] > arr[i])
8             ++slopes;
9     }
10
11     return slopes;
12 }
```

Да се докаже, че при подаден масив от цели числа `arr` със размер `n`, функцията `num_slopes(arr, n)` връща броят на максималните ненамаляващи подмасиви на `arr`.

Задача 2. Даден е следният алгоритъм:

```
1 int kadane(int *arr, int n)
2 {
3     int m = arr[0], curr_m = arr[0];
4
5     for (int i = 1; i < n; ++i)
6     {
7         if (arr[i] + curr_m > arr[i])
8             curr_m += arr[i];
9         else
10            curr_m = arr[i];
11
12        if (m < curr_m)
13            m = curr_m;
14    }
15
16    return m;
17 }
```

Какво връща `kadane(arr, n)`, където `arr` е масив от цели числа с дължина `n`? Обосновете отговора си.

Задача 3. Даден е следният алгоритъм:

```
1 int find_majority(int *arr, int n)
2 {
3     int maj = arr[0];
4     int cnt = 1;
5
6     for (int i = 1; i < n; ++i)
7     {
8         if (cnt == 0)
9         {
10            maj = arr[i];
11            cnt = 1;
12        }
13        else
14            cnt += (arr[i] == maj ? 1 : -1);
15    }
16
17    return maj;
18 }
```

Да се докаже, че при подаден масив от цели числа `arr` със размер `n`, в който има елемент с повече от  $\lfloor \frac{n}{2} \rfloor$  срещания, функцията `find_majority(arr, n)` ще върне точно този елемент.